ANLP Assignment 1 2021  
Marked anonymously: do not add your name(s) or ID numbers.

Use this template file for your solutions. Please start each question on a new page (as we have done here), do not remove the pagebreaks. Different questions may be marked by different markers, so your answers should be self-contained.

Don’t forget to copy your code for questions 1-5 into the Appendix of this file and to save your final version, then convert to .pdf for submission.

# Preprocessing each line (10 marks)

#We convert set to string for O(1) lookup

validCharacters = set("1234567890abcdefghijklmnopqrstuvwxyz. ")

#Process each line and add start/ending symbols

def preprocess\_line(line):

    #Add start-sentence symbol

    valid = "##"

    #Because each line ends with newline symbol, we simply delete them

    line=line.replace("\n","")

    #.lower() automatically changes all capitals to lower.

    for i in line.lower():

        if i in validCharacters:

            if i in set("1234567890"):

                valid += "0"

            else:

                valid += i

    #add end-sentence symbol

    valid+="#"

return valid

# Examining a pre-trained model (10 marks)

There are 30 possible characters including the start/end sentence symbol in this language system, so there should be 27000 possible trigrams. However, some trigrams are logically impossible (e.g., “a##” and “a#a”), so there are 26100 trigrams left, which are exactly contained in the pre-trained model. Among these trigrams, some are factually impossible, like “zxv”, which cannot be found in a dictionary (but admittedly, these trigrams may appear in some academic terms).

As every possible trigram in the model has a positive probability, and many of these trigrams (especially those “factually impossible” trigrams) have the same probability, we believe that Add-Alpha Smoothing or was likely the estimation method (e.g., backoff or interpolation) used, of which the probability 3.333e-02 is probably the alpha if the trigram doesn’t actually exist within the corpus (I do not agree with it, I think this number only indicates the number of possible characters. For example, if the condition bigram is “ ”, there are 30 possible trigrams but none of them are factual possible, so the smoothing result is α/30\*α=1/30). To further reduce uncertainty, we calculated the sum of conditional probabilities for each possible condition bigrams, and compared them with 1. All of them were close to 1 (with absolute tolerance 0.001), which indicated a naïve backoff impossible. Probably an advanced backoff method is used.

# Implementing a model: description and example probabilities (35 marks)

## Model description

Inspired by the pre-trained model, we introduced all logically possible trigrams to our model, and conducted a Add-One smoothing. We then estimated the conditional probability of each trigram using maximum likelihood method. Like the pre-trained model, we saved the results in a dictionary, with trigram as the key and conditional probability as the value.

## Model excerpt

Because most intuitive trigram given history “ng” is “ing” (the continuous tense of verbs ends with this trigram), we hypothesized that the most possible next character is a space. The model results supported our hypothesis, the conditional probability p(“ ”|“ng”) obviously dominated.

# Generating from models (15 marks)

To generate random output sequences, we first created a dictionary for each model, whose key was the history (the conditional bigram), and value was a list containing the possible next character and the corresponding conditional history. The following pseudocode shows how the sequence was then generated.

**Function** generate\_from\_LM (sequence\_length, model\_dictionary) **returns** a random sequence

sequence ← {“#”}

current\_length ← 0

current\_end ← {“#”}

**Loop**

**If** current\_length equals to sequence\_length

sequence ← sequence replacing “#” with “/n”

**return** sequence

**else**

**if** current\_end is “#”

next ← random sample from model\_dictionary[“##”]

sequence ← sequence ∪next

**else**

history ← last two characters from sequence

next ← random sample from model\_dictionary[history]

sequence ← sequence ∪next

current\_length ← length of sequence excluding “#”

**end**

The outputs based on my model were many short “sentences”, but those based the pre-trained model were few long “sentences”. This was probably because the sentences in my model’s corresponding corpus were usually shorter than those in the pre-trained model’s corresponding corpus.

# Computing perplexity (15 marks)

The perplexity was 8.87, 22.52, 22.94 respectively, for the English, Spanish and Germany model. The test document was more likely to be an English document because of its smaller perplexity. In other words, the harmonic average conditional probability of each character given its history is larger assuming the document was written in English. If our prior belief was uniform, the posterior belief should prefer the hypothesis that this was a English document.

It is not enough to make a judgement if we only run the English model on a new test document and get its perplexity. All models are wrong, but some are better (adapted from “all models are wrong, but some are useful”). Judgements based on perplexity only makes sense if we have multiple candidate models.

# Extra question (15 marks)

# Appendix: your code

Include a verbatim copy of your code for questions 1-5 here. If you answered question 6, you do *not* need to include that code.

Replace this line with your code. (Make sure the spacing looks right. Ideally, use Courier or another fixed-width font, like this one. However, if you have trouble getting the code to look right, you may use screenshot images. Either way, make sure your code is legible.)